PAHSCTRL User's Guide

March 24, 2011

# Contents

# Chapter 1

# Introduction

The PAHS Control Toolbox is a Matlab toolbox designed to aid in the design of controllers for piece-wise affine hybrid systems. The main purpose is to synthesise controllers for systems with uncontrollable external inputs, e.g. fault tolerant controllers, by computing discrete game abstractions of hybrid systems. The toolbox implements the ideas presented in [Grunnet08a-ND], section 33, which is based on the [Habets06a], section 33 regarding control to facet of PAHSs. The main capabilities are centered around checking which facets of a polytope an affine system can be prevented from leaving through, *blocking*, to which degree the system can be forced out a given facet, *controllability*, and by building on this basic capability compute discrete abstractions which describe the system dynamics as a discrete game. This is the basis for creating discrete game abstractions of PAHS, which tools such as UppAal[1] can synthesise solutions for. The PAHS Control Toolbox is capable converting the discrete game abstraction and Reach/Avoid/Stay requirements to UppAal syntax and if feasible find a winning strategy for the game. The winning strategy is then refined to piecewise affine control laws on the PAHS and a simulator based on the `trap_101` ODE solver is included. In summary: Given a piecewise-affine hybrid system with external events and a reach/avoid/stay specification the toolbox can automatically find a control strategy.

## 1.1 Functionality

The PAHS toolbox `V1.1` supports the following features

- Determine control to facet properties of affine systems on polytopes
  - Autodetection of simplices for faster computations
- Computing discrete equivalents of affine systems on polytopes
  - Visualising discrete equivalents (2D only)
- Computing discrete game abstractions of PAHS
  - Support for Matlab parallel computing (linear speedup)
- Converting Init, Goal and Avoid sets to specifications on a discrete abstraction
- Finding winning strategies for discrete abstractions using UppAal
- Refining strategies to affine control laws on the polytopes
  - User defineable optimisation functions
  - Montecarlo simulation of controlled affine systems on polytopes
  - Plotting Montecarlo simulations (2D only)
- Simulating controlled PAHS
  - Plotting PAHS and simulations (2D only)

## 1.2 Restrictions

There are a number of restrictions on the PAHS

- All resets are identity
- No guards on external events

[1] `http://www.uppaal.com`

## 1.3 Known bugs and regressions

This is a list of known bugs that should be corrected in future releases

- Controller generation does not support controllable external events
- New method for plotting monte carlo simulations and discrete equivalents does not work for 3D

## 1.4 Planned features

The following features are planned for future realeases (if any)

- Support for affine reset maps on external events
- Support guards on external events (unions of polytopes)
- Support for a Runge-Kutta based solver in the simulator
- Parallelisation of refinement function
- Helper functions for creating PAHS
- Yalmip support (initial tests shows its slower for small problems)
- Montecarlo simulation of controlled PAHS

## 1.5 Licensing

The PAHS toolbox as licensed under the GPL V3[2]. The toolbox includes sligthtly modified versions of files from the `RANDOM_DATA` library authored by John Burkhardt and licensed under the LGPL[3]. These can be found in the `burkhardt` subdirectory.

[2] `http://www.gnu.org/licenses/gpl.html`
[3] `http://www.gnu.org/licenses/lgpl.html`

# Chapter 2

# Quick Example

To give a feel of what can be done using the toolbox a quick example of how a discrete equivalent can be found for an affine system on a simplex.

Finding a discrete equivalent

```
%First define the state space simplex
s.dim=2;
s.vertices={[2;1],[1;2],[1,1]};
s=makesimplex(s);

%Then the system
s.dynamics.A=[0 1; 0 -0.3];
s.dynamics.B=[0;1];
s.dynamics.C=[0;0];

%Now the input polytope (in this case it is a simplex)
s.input.dim=1;
s.input.vertices={1,-1};
s.input=makesimplex(s.input);

%Prepare the simplex for computations
s=preparesimplex(s);

%Find a discrete equivalent
de=finddiscreteeq(s);

%Plot the result
plotdisceq(s,de);
```

The example above is a model of a block sliding on a 1-D surface with viscous friction. The input is the force applied on the block by e.g. a rod. The following example shows how controllers for a PAHS can generated.

Computing a control strategy for a PAHS

```
%First define the pahs.
%This PAHS is included in the toolbox.

pahs=demopahs2();

%Compute the discrete abstraction
dgabs=finddiscgame(pahs);

%Define the control requirements
req=demoreq2();

%Convert the requirements to a specification on the discrete abstraction
spec=req2spec(pahs,req)

%Find a winning strategy
```

```
[win,sol]=finddgsol(dgabs,spec);

%Is there a winning strategy
if(win)
    disp('Winning strategy found!');

    %Compute control laws based on the strategy
    ctcat=sol2ctrl(pahs,dgabs,sol);

    %Parameters for the simulation:

    %Initial location and mode
    l0=req.init{1}.loc;
    m0=req.init{1}.modes(1);

    %Initial state
    x0=mean(horzcat(pahs.mode{m0}.loc{l0}.s.vertices{:}),2);

    %Set up the occurrence of events
    events{1}.time=3;
    events{1}.mode=2;

    %Simulation time
    simtime=1000;

    %Timestep
    step=.5;

    %Simulate the controlled pahs
    [x,t,m,l]=simpahs(pahs,ctcat,x0,l0,m0,simtime,events,step);

    %Plot the results
    plotpahs(pahs,spec,x,t,events,m0);
else
    disp('No winning strategy found!')
end
```

The above example can be found as `example2.m` in the demo directory of the toolbox.

# Chapter 3

# Download and Installation

## 3.1 Requirements

To use the toolbox Matlab and the LMI toolbox is required. All scripts have been tested with Matlab version 7.5.0.338 (R2007b). Furthermore an openGL capable graphics card is needed to get full use of the plotting functions. To solve the discrete game abstractions UppAal Tiga is needed and to be able to simulate controlled PAHS the ODE solvers for hybrid systems authored by Prof. James Taylor are needed .

## 3.2 Download

The toolbox is available in zipped format pahsctrl (V1.1)[1] and the User's guide is available in pdf format[2]. It includes files from John Burkhardts RANDOM_DATA[3] library slightly modified to use Matlabs internal random generator. You also need Uppaal TIGA[4] and the hybrid systems ODE solvers[5].

## 3.3 Install

For easiest use of the toolbox make sure you have write permissions to the Matlab path-file. Else you have to rerun the install every time you start Matlab.

1. Unzip the software to a suitable location
2. Run the `install/install.m` file from Matlab
3. Run `install_check.m` in the tests-directory to test the installation

The installer assumes that you have the hybrid system ODE solvers in your Matlab path. If not a warning will be issued.

## 3.4 Changelog

v1.1

- Feature: Plotting simulations of 2D pahs
- Feature: Support differing partiton of modes
- Feature: Autodetection of Linux/Unix
- Feature: Install Scripts includes UppAal configuration
- Feature: Sanity Checks in spec2uppaal
- Bugfix: Fix datastructure abuse in findctrl
- Bugfix: Re-enable plotting of polytopes and MonteCarlo simulations (now 2D only)

v1.0 (Internal Release)

[1] `pahsctrl.zip`
[2] `pahsctrl-guide.pdf`
[3] `https://people.scs.fsu.edu/~burkardt/m_src/random_data/random_data.html`
[4] `http://www.cs.aau.dk/~adavid/tiga/`
[5] `http://www.ee.unb.ca/jtaylor/HS_software.html`

- Feature: Finding winning strategies to discrete abstraction using UppAal
- Feature: Conversion of reach/avoid/stay requirements to discret abstraction specification
- Feature: Refinement of controllers based on winning strategy
- Feature: Simulation of controlled pahs
- Bugfixes: Numerous

v0.95 (Internal Release)

- Feature: Added definition of a PAHS based on the simplex definitions
- Feature: Computation of discrete games
- Feature: Multithread support
- Bugfixes: To many to list

v0.91

- Bugfix: Correct the handling of config options
- Bugfix: Make sure subsets of blockable sets with no discrete equivalents are also searched.
- Bugfix: Make sure to check for discrete equivalent with no facets blocked if no other discrete equivalents can be found.
- Bugfix: Also Calculate control laws for no desired exit facet
- Bugfix: Avoid duplicate discrete equivalents
- Feature: Add the posibility for user defined test vectors for fixed point tests
- Feature: Cosmetic changes in plot function

V0.9 First Internal Release

# Chapter 4

# Contact Information

The PAHS Control toolbox is developed by Jacob D. Grunnet[1] from Automation and Control[2] and CISS[3] at Aalborg University. Both feature requests and bug reports are very much appreciated and anyone using the toolbox is encouraged to contact Jacob[4]. The toolbox is a part of Jacob's PhD-project.

[1] http://www.control.aau.dk/~grnnet
[2] http://www.es.aau.dk/control
[3] http://www.ciss.dk
[4] grunnet@es.aau.dk

# Chapter 5

# Getting Started

The first thing to do is to download, section 3 and install the toolbox. The toolbox makes extensive use of the `LMI toolbox`, which is part of Matlab's `Robust Control toolbox` so make sure that it is installed as well.

## 5.1 Testing the installation

To make sure the toolbox has been installed correctly execute `install_check` in the tests directory. This should return `Install Succesfull` or hopefully a usefull hint if something fails

# Chapter 6

# Creating a Polytope

To be able to compute discrete equivalents the definition of the problem needs to be put on a form understandable by the toolbox. To this end a polytope data structure is defined, with three basic elements:

- Polytopic state space
- Affine dynamics
- Polytopic input space

Several examples are included in the `demo` subdirectory as `demosimplex*.m`/`demopolytope*.m` and a small example is shown below. Notice that the toolbox initially only supported simplices and this carries over in some of the naming conventions.

Example polytope creation (simplex)

```
%The dimension of the state space
s.dim=3;

%The vertices
s.vertices={[2;2;1],[1;2;1],[1;1;1],[1.3;1.3;2]};

%Helper function simplicial state/input space. If you want to create a
%polytope follow the steps shown for the input
s=makesimplex(s);

%The system dynamics
s.dynamics.A=[0 1 0; 0 0 1; 0 0 0];
s.dynamics.B=[0 1; 1 0;0 0 ];
s.dynamics.C=[0;0;0];

%The input space is a 2 dimensional polytope
s.input.dim=2;
s.input.vertices={[1;0.5],-[1;0.5],[-1;0.5],[1;-0.5]};

%We can't use makesimplex on polytopes so it is created manually
%The vertices of facet 1
s.input.facet{1}.vertices={1,3};

%A vertice not belonging to facet 1
s.input.facet{1}.opposit=4;

s.input.facet{2}.vertices={1,4};
s.input.facet{2}.opposit=2;
s.input.facet{3}.vertices={2,3};
s.input.facet{3}.opposit=1;
s.input.facet{4}.vertices={2,4};
s.input.facet{4}.opposit=3;

%Prepare the polytope
```

```
s=preparepolytope(s);
```

When the computations on the simplex are done, some space can be saved by using the function `s=cleansimplex(s)`.

# Chapter 7

# Finding blockable facet sets

The conerstone of this toolbox is to be able to check whether a set of facets are blockable. A number of functions are provided to perform the test and search for blockable facet sets. It is easy to test if a facet set is blockable

Test a facet set for blockability

```
%s is a prepared simplex

%a facet set
fset=[1 3];

%Is it blockable
if(isblockable(s,fset))
 disp 'The facet set is blockable'
else
 disp 'The facet set is not blockable'
end
```

## 7.1 Searching for blockable facet sets

Often there is no knowledge of which facet sets are blockable and it is thus necessary to search for blockable sets. The toolbox provides two functions for this purpose. The functions are optimised such that not all permutations of facets needs to be checked.

Find blockable facet sets

```
%s is a prepared simplex

%finds a blockable set with as many facets as possible
bset=findblockable(s);

%finds all blockable sets of s
bsetlist=allblockable(s)
```

# Chapter 8

# Determining Controllability

When a blockable facet set is found it is usefull to check whether a given facet in the set is controllable. In this context controllability of a facet is defined as the ability to block all other facets in the blockset while ensuring that the vector field points out of the controllable facet. Also it must be ensured that there are no fixed points in the simplex. Notice that in the current implementation, if a facet is found uncontrollable it does not mean that it is not possible to control, but rather that the algorithm couldn't positively confirm that the facet is controllable. An example of how to use the two functions supplied to determine controllability

Is the blockset controllable

```
%s is a prepared simplex and bset is a blockable facet set

%is facet 2 controllable
if(iscontrollable(s,bset,2))
  disp 'Facet 2 is controllable'
else
  disp 'Facet 2 is not controllable'
end

%Which facets of the blockset are controllable and which
%testvectors was it proven with.
[cset,testVecs]=findcontrollable(s,blockset)
```

# Chapter 9

# Computing Discrete Equivalents

Finding blockable facet sets and determining witch facets are controllable is exactly the information needed when computing discrete equivalents. As such the discrete equivalents can be found using the functions already described, but for the sake of usability a number of helper functions are supplied.

Getting a discrete equivalent of a blockset

```
%s is a prepared simplex and bset is a blockable facet set
de=getdiscreteeq(s,bset);

%if not all facets of a discrete equivalent are controllable there
%might exist more than one de for a blockset
subdes=findsubde(s,de);
```

## 9.1 Finding discrete equivalents directly from the simplex

Most of the time the only reason to find blockable facet sets is to compute discrete equivalents. To speed this process up a couple of functions have been defined.

Search for discrete equivalents

```
%s is a prepared simplex

%searching for a discrete equivalent without a blockset
de=finddiscreteeq(s);

%finding all discrete equivalents of a simplex
delist=alldiscreteeq(s);
```

The de data structure is described in the function reference, section 18.

# Chapter 10

# Refinement to Control Laws

When a desired solution has been found to the discrete equivalent an actual control law needs to be computed. The function `findctrl(..)` is designed to automate this refinement procedure. The controller found is of the form $u = kx + g$ and is calculated based on the minimisation of a linear objective function. Currently two objective functions are implemented and it is possible to specify custom objective functions.

Getting a control law

```
%s is a prepared simplex with de being a discrete equivalent of s
%and findminsqinput is the objective function

%Find the control law for exiting through facet 2 with
ct=findctrl(s,de,2,@findminsqinput);

%extract the control law constants
k=ct.k;
g=ct.g;
```

## 10.1 Objective functions

Two objective functions are supplied with the toolbox

**`findminsqinput()`** Finds a control law for the discrete equivalent while minimising $J = \sum u_i^T u_i$ where $u_i$ is the control input at vertex $i$. This effectivly minimises the input effort at any given point in time. Notice that this does **n**ot translate to min$\int u^T u dt$.

**`findbesteffortinput(s,de,outfacet)`** Finds the best effort input in the sense that it tries to maximise the flow in the direction of the desired exit facet while minimising the flow in the direction of the uncontrollable facets.

For details on how to create custom objective functions in the [reference.html

# Chapter 11

# Plotting the Result

When a discrete equivalent has been found and/or a control law has been computed it is nice to be able to visualise the result. For this purpose two plot functions are included in the toolbox.

**`plotdisceq`** Plots discrete equivalents of a simplex with state spaces of dimension 2. The facet are color coded according to if they are blocked, opened or uncotrollable.

**`plotmcsim`** Performs a monte-carlo simulation of a control law applied to an affine system. The initial conditions for the simulations are uniformly distributed in the state space polytope.

Plotting and Simulation

```
%de is a discrete equivalent of the prepared simplex s and ct is the control
    parameters for exiting facet 2.

%plot the discrete equivalent marking facet 2 as exit
h=plotdisceq(s,de,2);

%plot a monte-carlo simulation of the control to exit facet 2
%Note that this is plottet on top of the discrete equivalent
plotmcsim(s,ct,h);
```

# Chapter 12

# Creating a PAHS

In order for the toolbox to synthesis controllers for a PAHS it needs to be defined. The PAHS specification contains a set of polytopes for each mode and a description of the transitions between the modes. Several examples are included in the `demo` subdirectory as `demopahs*.m` and a an example is shown below.

Example pahs creation

```
%It is assumed that s1 through s3 are fully specified polytopes
%including dynamics and input sets, while s4 to s6 are polytopes with
%no input set defined

%e is defined as an empty set for convinience
e={};

%Create mode 1

%First polytope
mode1.loc{1}.s=s1;

%Specify which polytopes it borders. The bordering polytopes are
listed in facet order and if a facet borders unpartitioned space an
empty set is inserted.
mode1.loc{1}.bord={2,e,3,e};

%Ditto for polytope 2 and 3
mode1.loc{2}.s=s2;
mode1.loc{2}.bord={e,e,1,e};

mode1.loc{3}.s=s3;
mode1.loc{3}.bord={1,e,e,e};

%Add an uncontrollable transition to mode 2
mode1.trans{1}.dest=2;
mode1.trans{1}.ctrb=false;

%Create mode 2, input2 is the common input polytope for polytopes 4 to 6

%Set common input polytope
mode2.input=input2;

%The same could be done for if all polytopes in a mode has the same dynamics
%mode2.dynamics=dyn2;

mode2.loc{1}.s=s4;
mode2.loc{1}.bord={e,2,3};

mode2.loc{2}.s=s5;
mode2.loc{2}.bord={1,e,e,3};
```

```
mode2.loc{3}.s=s6;
mode2.loc{3}.bord={1,e,e,2};

%no transitions from mode 2
mode2.trans={};

%Create the pahs
pahs.mode{1}=mode1;
pahs.mode{2}=mode2;

%Does the modes have the same partition, in this case no
pahs.samepart=false;
```

# Chapter 13

# Specifying Control Requirements

The control requirements supported by the toolbox is of the form of reachavoidstay specifications i.e. starting in *init* reach and stay in *goal* while never entering *avoid*. The toolbox supports specifying *goal*, *init*, and *avoid* sets as unions of polytopes or alternatly directly as locations of the discrete abstraction.

Specifying Requirements

```
%The polytopes a constructed as with the pahs polytopes. This example uses a
    simplex.
p1.vertices{[1;1],[0;0],[0;1]}
p1=makesimplex(p1)

%The first init polytope is specified in both modes 1 and 2
req.init{1}.modes=[1 2];
req.init{1}.poly=p1;

%The avoid set is only in mode 2
req.avoid{1}.modes=2;
req.avoid{1}.poly=p2;
req.avoid{2}.modes=2;
req.avoid{2}.poly=p3;

%The goal set specified in both modes
req.goal{1}.modes=2;
req.goal{1}.poly=p4;

%The PAHS the requirements are made on
pahs=demopahs();

%Convert the requirements to specifications on the discrete abstraction
spec=req2spec(pahs,req);

%It is possible to add specifications on the discrete abstraction.
%Instead of polytopes the locations of the pahs/discrete abstraction
%are specified directly directly.
spec.avoid{end+1}.modes=1;
spec.avoid{end}.loc=3;
```

# Chapter 14

# Finding Winning Strategies based on Discrete Abstractions

Computing a discrete abstraction is simple once a PAHS has been defined, section 12. With the discrete game computed and corresponding specification a winning strategy can be computed using UppAal.

Finding a winning strategy

```
%Get the pahs
pahs=demopahs2();

%Compute the discrete abstraction
dgabs=finddiscgame(pahs);

%Get the requirement and convert them to a specification
req=demoreq2();
spec=req2spec(pahs,req);

%Try to find a winning strategy using UppAal
[win,sol]=finddgsol(dgabs,spec);

%Check if a winning strategy was found
if(win)
    disp('Winning')
else
    disp('Loosing')
end
```

# Chapter 15

# Computing a Controller Catalogue

When a winning strategy has been found to a discrete game, the solution needs to be refined to control laws of the form $u = Kx + g$. One control law is computed for each polytope using linear optimisation., section 31

Computing controllers

```
%Find a winning strategy
pahs=demopahs2();
dgabs=finddiscgame(pahs);
req=demoreq2();
spec=req2spec(pahs,req);
[win,sol]=finddgsol(dgabs,spec);

%Check if a winning strategy was found
if(win)
    disp('Winning')
   %Compute the controller catalogue
    ctcat=sol2ctrl(pahs,dgabs,sol);
else
    disp('Loosing')
end
```

# Chapter 16

# Simulate a Controlled PAHS

The simulator included in the toolbox uses a 2nd order trapezoid integration method especially taylored to detect when a guard has been crossed. In order to perform a simulation the user needs to specify initial mode,location and state as well as the timing of external events. The output of a simulation is:

- $x$: A vector of state values
- $t$: A timevector
- $l$: A vector of locations
- $m$: A vector of modes

Simulating a Control Strategy

```
%Compute a controller catalogue
pahs=demopahs2();
dgabs=finddiscgame(pahs);
req=demoreq2();
spec=req2spec(pahs,req);
[win,sol]=finddgsol(dgabs,spec);
if(win)
    disp('Winning')
    ctcat=sol2ctrl(pahs,dgabs,sol);

   %Setup the simulation
    %Initial location and mode
     l0=req.init{1}.loc;
     m0=req.init{1}.modes(1);

     %Initial state
     x0=mean(horzcat(pahs.mode{m0}.loc{l0}.s.vertices{:}),2);

     %Set up the occurrence of events
     events{1}.time=3;
     events{1}.mode=2;

     %Simulation time
     simtime=1000;

     %Timestep
     step=.5;

     %Simulate the controlled pahs
     [x,t,m,l]=simpahs(pahs,ctcat,x0,l0,m0,simtime,events,step);

end
```

# Chapter 17

# Plotting PAHS and Simulations

In order to visualise the results of the control synthesis via discrete game abstraction a plotting function has been included for systems with 2 dimensional dynamics. The plotting function plots each modes partition and overlays one or more of the requirements, discrete game specification or the results of a simulation.

Plotting PAHS

```
%Compute a controller and simulate
pahs=demopahs2();
dgabs=finddiscgame(pahs);
req=demoreq2();
spec=req2spec(pahs,req);
[win,sol]=finddgsol(dgabs,spec);
if(win)
    disp('Winning')
    ctcat=sol2ctrl(pahs,dgabs,sol);

    l0=req.init{1}.loc;
    m0=req.init{1}.modes(1);
    x0=mean(horzcat(pahs.mode{m0}.loc{l0}.s.vertices{:}),2);

    events{1}.time=3;
    events{1}.mode=2;

    simtime=1000;
    step=.5;

    [x,t,m,l]=simpahs(pahs,ctcat,x0,l0,m0,simtime,events,step);


   %Plot the pahs with requirements
    plotpahs(pahs,req);

   %Plot the pahs with specifications
    plotpahs(pahs,{},spec);

   %plot the pahs with a simulation and requirements overlayed
    plotpahs(pahs,req,x,t,events,m0);

     %plot the pahs with a simulation and specification overlayed
    plotpahs(pahs,{},spec,x,t,events,m0);

end
```

# Chapter 18

# Data structures

The PAHS Control toolbox uses a few basic datastructures. The data structures describe:

- Affine system on a polytope/simplex
- Hybrid game automata (non-deterministic PAHS)
- Discrete Equivalents of affine systems
- Discrete game abstractions of HGA
- Requirements and specifications for PAHS and discrete games
- Control catalogue implementing a winning strategy

Polytope with affine system

```
% s is the polytope
s.dim=N % The dimension of the statespace

s.vertices={#$v_1,\ldots,v_{l}$#} % Lists the #$l$# vertices of the polytope
%#$v_i$ are $\mathbb{R}^N$# vectors denominating the vertices of the state
    space polytope

s.facet{#$j$#}.vertices={#$j_1,\ldots,j_p$#} % Specifying the facets as a
    function of the vertices
%#$j=1\ldots p$# denotes a facet and #$j_i$# are indices into the list of
    vertices,
% indicating the vertices  of facet #$j$#

s.facet{#$j$#}.opposit=#$j_{o}$# %A vertex opposit the facet
% #$j_{o}$ is a vertex opposit facet $j$ i.e. any vertex of s not in
% s.facet{#$j$#}.

s.input % A substructure describing the polytopic input space for s
s.input.dim=m % The dimension of the input space

s.input.vertices={#$u_1$,\ldots $u_p$#} % The vertices of the input space
% # $u_i$ are $\mathbb{R}^m$ vectors denoting the vertices of the input space#

s.input.facet{#$j$#}.vertices={#$j_1,\ldots ,j_N$#}; % A facet of the input
    space
s.input.facet{#$j$#}.opposit=#$j_{N+1}$# % A vertex not in facet j

s.dynamics % Sub structure describing the dynamics acting on s
% #Dynamics are on the form $\dot{x}=Ax+Bu+C$#

s.dynamics.A=#$A\in \mathbb{R}^{N\times N}$# % The autonomous dynamics
s.dynamics.B=#$B\in \mathbb{R}^{N\times m}$# % The input dynamics
s.dynamics.C=#$C\in \mathbb{R}^{N}$# % The affine dynamics
```

Piecewise-Affine Hybrid System

```
%PAHS is a Piecewise-Affine Hybrid System

%A PAHS consists of a number of modes
pahs.mode{#$i$#}=#$m_i$# % The #$i'th$# mode of the PAHS

% Each mode has a number of substructures
mode.dynamics=dyn %Mode wide dynamics (not mandatory)
% Common dynamics shared between all polytopes of the mode

mode.input=input    %Mode wide input    (not mandatory)
% Common input shared between all polytopes of the mode

%Cell Array of locations
mode.loc{#$j$#}=#$q_{i,j}$# % is the #$j'th$# location of the #$i'th$# mode

%each location, $q_{i,j}$ (loc) is described as:
loc{#$j$#}.s  % an affine system on a polytope
loc{#$j$#}.bord=[#$k_1,\ldots,k_p$#]
% is an array of indices, one for each facet of loc.s,
% denoting neigboring locations to that facet.
% If empty there is no defined neighbor to the facet.

mode.trans{#$t$#} % Cell array of transitions starting from this mode
% Each transitions is described by its destination and controllability
trans{#$t$#}.dest=#$t_dst$#  %Destination mode #$i$# of the transition
trans{#$t$#}.ctrb=#$t_c$#   % Boolean describing the controllability of the
    transition, if true the transition is controllable.

pahs.samepart  %Boolean hint set to true if identical partitions is used in all
    modes
```

The discrete equivalent data structure

```
% de is the discrete equivalent
de.blockset=#$B \in \mathbb{N}^k$# % The blockable set
% #$B$# is a vector of #$k$# facet indices

de.exits=#$E \in \mathbb{N}^k$# % Possible exit facet vector
% #$E$# is a vector of #$k$# facet indices

de.exit{j}.testVec=#$tv \in R^N$# % The test vector used for facet j

de.block.testVec=#$tv \in R^N$# % Test vector used for the case of all facets
    blocked
```

The discrete abstraction data structure

```
% dgabs is a discrete game abstraction

dgabs.maxfacet=#$mf$# % this is the maximum number of facets of in any of the
    modes

mode.trans{#$t$#} % Cell array of transitions starting from this mode
% Each transitions is described by its destination and controllability
trans{#$t$#}.dest=#$t_dst$#  %Destination mode #$i$# of the transition
trans{#$t$#}.ctrb=#$t_c$#   % Boolean describing the controllability of the
    transition, if true the transition is controllable.

dgabs.mode{#$i$#}=#$dg_i$# %is the abstraction of mode #$i$#
```

```
mode.da{#$j$#}=#$d_{i,j}$# % is the abstraction of location #$j$# in mode #$i$#.
da{#$j$#}.des{#$k$#}=de % is a cell array of discrete equivalents of the
    location
da{#$j$#}.numfacets=#$f$# % is the number of facets of the location
da{#$j$#}.bord=[#$k_1,\ldots,k_p$#] %bordering locations identical to the pahs
    definition
% is an array of indices, one for each facet of loc.s,
% denoting neigboring locations to that facet.
% If empty there is no defined neighbor to the facet.

da{#$j$#}.trans{#$t$#}=[#$l_1,\ldots,l_e$#] % is a cell array of arrays (one
    for each #$t$#)
% The array contains indices $l_k$ specifying which locations in the
    destination mode of #$t$#
% can be reached by #$t$# from the discrete abstraction of #$q_{i,j}$#.
```

Specifications can be generated from requirements using `req2spec.m`, but it is possible both to describe requirements as polytopes on the PAHS and directly on the discrete abstraction using the spec structure.

Requirements and Specifications

```
%req is a requirement specification and #$P_l\ l=1\ldots k$# are polytopes.

%initial set
req.init{#$i$#} % Cell array of initial set polytopes. Each cell contains a
    polytope
 init{#$i$#}.poly=#$p_1$# % Polytope describing an initial set
 init{#$i$#}.modes=[#$m_1\ldots m_j$#] % An array of modes the polytope is
     valid in

%goal set
req.goal{#$i$#} % Cell array of goal set polytopes. Each cell contains a
    polytope
 goal{#$i$#}.poly=#$p_2$# % Polytope describing a goal set
 goal{#$i$#}.modes=[#$m_1\ldots m_j$#] % An array of modes the polytope is
     valid in

%avoid set
req.avoid{#$i$#} % Cell array of avoid set polytopes. Each cell contains a
    polytope
 avoid{#$i$#}.poly=#$p_1$# % Polytope describing an avoid set
 avoid{#$i$#}.modes=[#$m_1\ldots m_j$#] % An array of modes the polytope is
     valid in

%The specification on the dgabs is very similar to the requirements
%Here only the initial set specification is shown.

%Initial set
req.init{#$i$#} % Cell array of inital sets
 init{#$i$#}.loc=#$l_1$# % An initial location
 init{#$i$#}.modes=[#$m_1\ldots m_j$#] % An array of modes for the initial
     location
 %NB: Having more than one mode in the array mostly makes sense for problems
     where the
 %modes share a partition.
```

# Chapter 19

# Functions to Determine Blockability

isblockable

```
% ISBLOCKABLE checks if a set of facets of a polytope can be blocked
%
% Usage: BLOCKABLE=ISBLOCKABLE(S,FACETSET)
%
% S is a prepared polytope
% FACETSET is a vector with entries listing the facet numbers e.g. [1 3]
% referes to facet 1 and 3 of S. If no FACETSET is supplied it is tested
% whether all facets of S can be blockad simultaniously.
%
% BLOCKABLE is true if the facet set is blockable
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 s=isblockable(s);

% See also FINDBLOCKABLE, ISCONTROLLABLE, FINDCONTROLLABLE, and GETDISCRETEEQ
```

findblockable

```
% FINDBLOCKABLE searches for a blockable facet set on a simplex.
%
% Usage: [BLOCKSET,OUTQUEUE]=FINDBLOCKABLE(S,QUEUE,BLIST)
%
% S is a prepared polytope.
% QUEUE (optional) is a cell array of facet sets. Each facet set is a vector
    with
% entries listing the facet numbers e.g. [1 3] referes to facet 1 and 3 of s.
% BLIST (optional) is a cell array of facet sets which are known to be
% blockable on the simplex.
%
% OUTQUEUE is the remaining queue of facet sets
% BLOCKSET is the first blockable set found, null if none where found
%
% The algorithms is optimised by removing, from the queue, any subsets
% of blocksets which has already been tested as per the optional argument
% blist.
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 blockset=findblockable(s);
```

```
% See also ISBLOCKABLE
```

allblockable

```
% ALLBLOCKABLE finds all blockable facet sets of a polytope.
%
% Usage BLIST=ALLBLOCKABLE(S,QUEUE)
%
% S is a prepared polytope
% QUEUE is an optional argument limiting the search to the facet sets in q
% and their sub sets.
%
% BLIST is a cell array of blockable sets.
%
% Example
%
% s=demosimplex();
% s=preparepolytope(s);
% blist=allblockable(s);
%
% See also ISBLOCKABLE, FINDBLOCKABLE and PREPAREPOLYTOPE
```

# Chapter 20

# Functions Used to Determine Controllability

iscontrollable

```
% ISCONTROLLABLE checks for controllability using normals to facets as
% testvectors
%
% Usage [CONTROLLABLE,VEC]=ISCONTROLLABLE(S,BLOCKSET,FACET)
%
% S is the polytope
% BLOCKSET is a blockable set of facets. If facet is contained in the
% BLOCKSET no solution can be found.
% FACET is the facet to be checked for controllability. FACET being in
% BLOCKSET is contradictory and therefore returns false.
%
% CONTROLLABLE is a boolean, true if the facet is controllable.
% VEC is the vector in R^N for which controllability was proven.
% If there can be no fixed point in the simplex i.e. S.FPPOSSIBLE=FALSE then
% VEC is empty
%
% Notice that if this function returns false it does not mean that no
% control exists rendering the facet controllable, but only that one could
% not be found using this method.
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 blockset=findblockable(s);
 facet=blockset(1);
 blockset=blockset(2:end);
 controllable=iscontrollable(s,blockset,facet);

% See also ISBLOCKABLE and FINDCONTROLLABLE
```

findcontrollable

```
% FINDCONTROLLABLE checks a blockset for controllable facets
%
% Usage  [CSET,TVECS]=FINDCONTROLLABLE(S,BLOCKSET)
%
% S is the polytope
% BLOKSET is a blockable facet set on s
%
% CSET is a vector of booleans, coresponding to blockset. e.g. BLOCKSET =
% [4 5 7] and cSet=[0 1 0] means that facet 5 is controllable while 4 and 7
% are not.
% TVECS is a cell array of vectors used to test controllability.
%
```

```
% Example:

  s=demosimplex();
  s=preparepolytope(s);
  blockset=findblockable(s);
  cSet=findcontrollable(s,blockset);

% See also ISCONTROLLABLE and FINDBLOCKABLE
```

# Chapter 21

# Functions Used to Find Discrete equivalents

getdiscreteeq

```
% GETDISCRETEEQ computes a discrete equivalent of polytope with a blockset.
%
% Usage: DE=GETDISCRETEEQ(S,BLOCKSET)
%
% S is the polytope
% BLOCKSET is a facet set known to be blockable, see also ISBLOCKABLE
%
% DE is a discrete equivalent structure
%
% WARNING inputing a facet set which is not blockable results in undefined
% behaviour.
%
% The DE structure in detail:
%
% Blockability
% DE.BLOCKSET=[1 2 3] - The blockable set
%
% Controllability
% DE.EXITS=[1 3] - The possible exit facets
% DE.EXIT{3}.TESTVEC=[1 4 5] - The testvector controlability was proven with
% DE.BLOCK.TESTVEC=[1 4 5] - The testvector controlability was proven with
% for the case with all facets in the BLOCKSET blocked.
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 blockset=findblockable(s);
 de=getdiscreteeq(s,blockset)

% See also ISBLOCKABLE, FINDBLOCKABLE and FINDCONTROLLABLE
```

finddiscreteeq

```
% FINDDISCRETEEQ finds a discrete equivalent of a polytope.
%
% Finds a discrete equivalent of a polytope using a queue of facet sets as
% the starting point for searching for blockable sets.
%
% Usage: [DE,OUTQUEUE]=FINDDISCRETEEQ(S,QUEUE)
%
% S is a prepared polytope.
% QUEUE (optional) is a cell array of facet sets. Each facet set is a vector
    with
% entries listing the facet numbers e.g. [1 3] referes to facet 1 and 3 of
```

```
% S.
%
% DE is a structure discribing the discrete equivalent, see GETDISCERETEEQ
% OUTQUEUE is the remaining queue that can be searched for other discrete
% equivalents
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 de=finddiscreteeq(s);

% See also GETDISCRETEEQ AND ALLDISCRETEEQ
```

alldiscreteeq

```
% ALLDISCRETEEQ finds all discrete equivalents of a polytope
%
% Usage: DELIST=ALLDISCRETEEQ(S,QUEUE)
%
% S is the polytope
% QUEUE is an optional cell array of facet sets limiting the search to
% those sets and their sub sets.
%
% DELIST is a cell array of discrete equivalents in the form of de structs,
% see GETDISCRETEEQ
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 delist=alldiscreteeq(s);

% See also GETDISCRETEEQ, FINDDISCRETEEQ
```

findsubde

```
% FINDSUBDE finds all sub discrete equivalents of a discrete equivalent given
% on a a simplex
%
% Usage: SUBDES=FINDSUBDE(S,DEQUEUE)
%
% S is the polytope
% DEQUEUE is a cell array of discrete equivalents for which sub DEs needs
% to be found.
%
% SUBDES is a cell array of sub DEs which was found, represented by DE
% structs.
%
% Note: FINDSUBDE returns immidiately if PAHSconf.subde=false
%
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 de=finddiscreteeq(s);
 deq={de};
 subdes=findsubde(s,deq);

% See also FINDDISCRETEEQ
```

# Chapter 22

# Functions Used to Manipulate the Discrete Game

finddiscgame

```
% FINDDISCGAME computes a discrete game abstraction of a PAHS
%
% Usage DGABS=FINDDISCGAME(PAHS)
%
% PAHS is a Piecewise-Affine Hybrid System
% PAHS.MODE
%  MODE.DYNAMICS %Mode wide dynamics (not mandatory)
%  MODE.INPUT    %Mode wide input    (not mandatory)
%  MODE.LOC{}    %Cell Array of locations
%   LOC.S        %The simplex represented by the location
%   LOC.BORD[]   %Array of neigbouring simplices index coorsponds to facet
%  MODE.TRANS{}  %Transitions starting from this mode
%   TRANS.DEST   %Destination mode of the transition
%   TRANS.CTRB   %Controllability of the transition
%
% PAHS.SAMEPART  %Boolean hint set to true if identical partitions is used
%                  in all modes

% Example:

 pahs=demopahs();
 dgabs=finddiscgame(pahs);

% See also FINDDGSOL
```

finddgsol

```
% FINDDGSOL finds a winning strategy (if it exists) to a discrete game given a
    specification.
%
% Usage: [WIN,SOL]=FINDDGSOL(DGABS,SPEC)
%
% DGABS is a discrete game abstraction
% SPEC is a reach/avoid/stay specification
% WIN is a boolean, true if a winning strategy was found
% SOL is a solution implementing the winning strategy
%
% Example:

 pahs=demopahs();
 req=demoreq();
 spec=req2spec(pahs,req);
 dgabs=finddiscgame(pahs);
 [win,sol]=finddgsol(dgabs,spec)
```

```
% See also FINDDISCGAME and REQ2SPEC
```

req2spec

```
% Converts REACH/AVOID/STAY requirements defined on polytopes to a dgabs spec
%
% REQ.INIT %Initial set
%  INIT{i}.modes[]  %Vector of modes
%  INIT{i}.poly     %Polytope describing the set
%
% REQ.GOAL %Goal set
%  GOAL{i}.modes[]
%  GOAL{i}.poly     %Polytope describing the set
%
% REQ.AVOID %Avoid set
%  AVOID{i}.modes[]
%  AVOID{i}.poly     %Polytope describing the set
%
% Example:

 init.dim=2;
 init.vertices={[0;-1],[0.5;-1],[1;-0.5],[0;-0.5]};
 init.facet{1}.vertices={1,2};
 init.facet{1}.opposit=3;
 init.facet{2}.vertices={2,3};
 init.facet{2}.opposit=4;
 init.facet{3}.vertices={3,4};
 init.facet{3}.opposit=1;
 init.facet{4}.vertices={4,1};
 init.facet{4}.opposit=2;

 avoid.dim=2;
 avoid.vertices={[2.6;0.95],[2.95;0.95],[2.95;0.7],[2.85;0.7]};
 avoid.facet{1}.vertices={1,2};
 avoid.facet{1}.opposit=3;
 avoid.facet{2}.vertices={2,3};
 avoid.facet{2}.opposit=1;
 avoid.facet{3}.vertices={3,4};
 avoid.facet{3}.opposit=2;
 avoid.facet{4}.vertices={4,1};
 avoid.facet{4}.opposit=2;

 goal.dim=2;
 goal.vertices={[2;0.5],[3;0.5],[3;-0.5],[2;-.5]};
 goal.facet{1}.vertices={1,2};
 goal.facet{1}.opposit=3;
 goal.facet{2}.vertices={2,3};
 goal.facet{2}.opposit=1;
 goal.facet{3}.vertices={3,4};
 goal.facet{3}.opposit=2;
 goal.facet{4}.vertices={4,1};
 goal.facet{4}.opposit=2;

 req.init{1}.modes=[1 2];
 req.init{1}.poly=init;

 req.goal{1}.modes=[1 2];
 req.goal{1}.poly=goal;

 req.avoid{1}.modes=2;
 req.avoid{1}.poly=avoid;
```

```
  spec=req2spec(pahs,req);

% See also FINDDGSOL and PLOTPAHS
```

# Chapter 23

# Functions Used to Find Control Laws

findctrl

```
% FINDCTRL finds the control on a simplex that realises a discrete equivalent.
%
% Usage: CT=FINDCTRL(S,DE,OUTFACET,@OPTFUNC,ARGS);
%
% S is a prepared polytope
% DE is a discrete equivalent of s, which can be found using
% FINDDISCRETEEQ
% OUTFACET is the desired outputfacet, must be in the blockable set or be
% empty if the facet is to be left through an uncontrollable facet or stay
% inside.
% @OPTFUNC is a function-pointer to an optimisation function
%  See the documentation for instructions on creating optimisation functions.
%  Functions included in the toolbox are:
%  FINDBESTEFFORTINPUT(S,DE,OUTFACET)
%  FINDMINSQINPUT()
% ARGS are any arguments that must be supplied to the optimisation
% function.
%
% CT is the control realising the DE
%  CT.K=K - Control gain
%  CT.G=G - Affine control gain
%
% U=KX+G is the control law achiving the optimum values
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 de=finddiscreteeq(s);
 out=de.exits(1);
 ct=findctrl(s,de,out,@findminsqinput);

% See also FINDBESTEFFORTINPUT, FINDMINSQINPUT and FINDDISCRETEEQ
```

sol2ctrl

```
% SOL2CTRL refines a winning strategy on a discrete game abstraction to a
% controller catalogue
%
% Usage: CTCAT=SOL2CTRL(PAHS,DGABS,SOL)
%
% PAHS a piecewise-affine hybrid system
% DGABS a discrete game abstraction of PAHS
% SOL a solution to the DGABS obtained by FINDDGSOL
%
```

```
% CTCAT a catalogue of controllers
%  CTCAT.MODE{i}.LOC{J}.PREV{K} des to choose when previous polytope was K
%  CTCAT.MODE{i}.LOC{J}.DES controllers for the different discrete
%  equivalents
%  CTCAT.MODE{i}.LOC{J}.DES{L}.PREV{K} exit to choose when previous
%   polytope was K, if 0 block all blockable facets
%  CTCAT.MODE{i}.LOC{J}.DES{L}.EXIT{M}.K control parameter when desired
%   exit facet is M
%  CTCAT.MODE{i}.LOC{J}.DES{L}.EXIT{M}.G control parameter when desired
%   exit facet is M
%  CTCAT.MODE{i}.LOC{J}.DES{L}.BLOCK.K control parameter when all facet
%   must be blocked
%  CTCAT.MODE{i}.LOC{J}.DES{L}.BLOCK.G control parameter when all facet
%   must be blocked
%
% Example:

 pahs=demopahs2();
 dgabs=finddiscgame(pahs);
 req=demoreq2();
 spec=req2spec(pahs,req);
 [win,sol]=finddgsol(dgabs,spec);
 if(win)
    disp('Winning')
    ctcat=sol2ctrl(pahs,dgabs,sol);
 end

% See also SIMPAHS and FINDDGSOL
```

findminsqinput

```
% FINDMINSQINPUT optimisation function used for FINDCTRL.
% DO NOT call this function directly, see the documentation for details.
%
% Minimises J=SUM UI'*UI where UI denotes the input at facet I.
%
% Usage: As a parameter to the findctrl function.
% No extra arguments are needed for this function.
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 de=finddiscreteeq(s);
 out=de.exits(1);
 ct=findctrl(s,de,out,@findminsqinput);

% See also FINDCTRL, FINDBESTEFFORTINPUT and GETDISCRETEEQ
```

findbesteffortinput

```
% FINDBESTEFFORTINPUT optimisation function used for FINDCTRL.
% DO NOT call this function directly, see the documentation for details.
%
% Optimises the vectorfield of the controlled system to point towards the
% desired exit facet and away from uncontrollable facets.
%
% Usage: As a parameter to the findctrl function.
% The extra arguments that needs to be passed to this function is
% S,DE,OUTFACET.
%
```

```
% S is the polytope
% DE is the discrete equivalent
% OUTFACET is the facet which the controller should try and exit
% through
%
% Example:

  s=demosimplex();
  s=preparepolytope(s);
  de=finddiscreteeq(s);
  out=de.exits(1);
  ct=findctrl(s,de,out,@findbesteffortinput,s,de,out);

% See also FINDCTRL, FINDMINSQINPUT and GETDISCRETEEQ
```

# Chapter 24

# Plotting Functions

plotdisceq

```
% PLOTDISCEQ plots a discrete equivalent of a polytope using colours to mark
% the properties of the facet: Blocked = black, uncontrollable=yellow and
% desired exit facet = green.
%
% Usage H=PLOTDISCEQ(S,DE,OUT,CTRL,FHANDLE)
%
% S is the polytope being plottet
% DE is the discrete equivalent to be plottet
% OUT is an optional the intended exit facet
% CTRL is optional control on the polytope u=CTRL.K*x+CTRL.g
% FHANDLE is an optional handle to the figure to be plottet on
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
 de=finddiscreteeq(s);
 out=de.exits(1);
 ct=findctrl(s,de,out,@findminsqinput);
 h=plotdisceq(s,de,out,ct);

% See also GETDISCRETEEQ, PLOTMCSIM
```

plotmcsim

```
% PLOTCSIM plots Monte Carlo simulations on a polytope using the control
% law given by a discrete equivalent and desired exit facet.
%
% Usage: H=PLOTMCSIM(S,CTRL,FHANDLE,SIMS,TIME,SEED)
%
% S is the affine system on a polytope to be simulated
% CTRL.K and CTRL.G are the control gains used as u=Kx+G
% FHANDLE is an optional figure handle to draw on.
% The following parameters are optional but can also be set from the
% configuration file.
% SIMS is the number of simulations to perform
% TIME is the simulation time given as an interval, see ODE45
% SEED is parsed to the rng, see RAND
%
% H is the figure handle of the figure being drawn on
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);
```

```
 de=finddiscreteeq(s);
 out=de.exits(1);
 ct=findctrl(s,de,out,@findminsqinput);
 h=plotdisceq(s,de,out,ct);
 plotmcsim(s,ct,h);

% See also FINDCTRL, PLOTDISCEQ, FIGURE, RAND, ODE45
```

plotpahs

```
% PLOTPAHS plots a 2D pahs optionally including, requirements,
% specifications or simulation traces.
%
% Usage: [FIGS]=PLOTPAHS(PAHS,REQ,SPEC,X,T,EVENTS,M0)
%
% PAHS is the pahs to be plottet
% REQ is the requirements, if empty it will not be plottet
% SPEC (optional 1) is the specification on the corresponding discrete
    abstraction
% X (optional 2) is the trace of a simulation
% T (optional 2) is the time vector corresponding to the entries in X
% EVENTS (optional 2) is the event times, see SIMPAHS
% M0 (optional 2) is the initial mode
%
% FIGS is a cell array of figure handles for the figures being drawn on
%
% Example:

 pahs=demopahs2();
 dgabs=finddiscgame(pahs);
 req=demoreq2();
 spec=req2spec(pahs,req);
 [win,sol]=finddgsol(dgabs,spec);
 if(win)
     disp('Winning')
     ctcat=sol2ctrl(pahs,dgabs,sol);
     l0=req.init{1}.loc;
     m0=req.init{1}.modes(1);
     x0=mean(horzcat(pahs.mode{m0}.loc{l0}.s.vertices{:}),2);
     events{1}.time=3;
     events{1}.mode=2;
     simtime=1000;
     step=.5;
     [x,t,m,l]=simpahs(pahs,ctcat,x0,l0,m0,simtime,events,step);

     plotpahs(pahs,req,x,t,events,m0);
 end

% See also SIMPAHS and REQ2SPEC
```

# Chapter 25

# Simulation Functions

simpahs

```
% SIMPAHS simulates a controlled pahs using a trapezoid ODE solver
%
% Usage [X,T,M,L]=SIMPAHS(PAHS,CTCAT,X0,L0,M0,TIME,EVENTS,STEP)
%
% PAHS the pahs to be controlled
% CTCAT the controller catelog (SEE SOL2CTRL)
% X0 initial continous state
% L0 the initial location X0 belongs to
% M0 the initial mode
% TIME the simulation time
% EVENTS externally triggerd event times
%  EVENTS{i}.TIME=10 time of the event
%  EVENTS{i}.mode=2 the destination mode of the transition
%  Note that the time of event i must be grater than the time of event i-1
% STEP the step size used by the solver
%
% X a vector of states
% T the time corresponding to the state vector
% M a vector of modes
% L a vector of locations
%
% Example:

 pahs=demopahs2();
 dgabs=finddiscgame(pahs);
 req=demoreq2();
 spec=req2spec(pahs,req);
 [win,sol]=finddgsol(dgabs,spec);
 if(win)
     disp('Winning')
     ctcat=sol2ctrl(pahs,dgabs,sol);
     l0=req.init{1}.loc;
     m0=req.init{1}.modes(1);
     x0=mean(horzcat(pahs.mode{m0}.loc{l0}.s.vertices{:}),2);
     events{1}.time=3;
     events{1}.mode=2;
     simtime=1000;
     step=.5;
     [x,t,m,l]=simpahs(pahs,ctcat,x0,l0,m0,simtime,events,step);
 end

% See also PLOTPAHS and SOL2CTRL
```

# Chapter 26

# Utility Functions

preparesimplex

```
% PREPARESIMPLEX calculates normals and lmi's for a simplex with an affine
% system and an input polytope defined.
%
% Usage: S=PREPARESIMPLEX(S)
%
% S is the simplex to be prepared
%
% Example:

 s=demosimplex();
 s=preparesimplex(s);

% See also MAKESIMPLEX and CLEANSIMPLEX
```

preparepolytope

```
% PREPAREPOLYTOPE calculates normals and lmi's for a polytope with an affine
% system and an input polytope defined.
%
% Usage: S=PREPAREPOLYTOPE(S)
%
% S is the polytope to be prepared
%
% Example:

 s=demosimplex();
 s=preparepolytope(s);

% See also PREPARESIMPLEX
```

cleansimplex

```
% CLEANSIMPLEX reduces the memory footprint of a simples by removeing
% unnecessary data. Notice that this marks the simplex as not prepared.
%
% Usage: S=CLEANSIMPLEX(S)
%
% S is the simplex to clean
%
% Example:

 s=demosimplex()
 s=preparesimplex(s)
 %Do operations on the simplex
 s=cleansimplex(s);
```

```
  save s;

% See also PREPARESIMPLEX
```

makesimplex

```
% MAKESIMPLEX constructs a simplex from a list of vertices
%
% Usage:
%    S.DIM=N
%    S.VERTICES={V1,V2,V3};
%    S=MAKESIMPLEX(S)
%
% S.DIM is the dimension of the simplex
% VI is the I'th vertex of the simplex
%
% Notice that facet I will be defined by S.VERTICES/VI.
%
% Example:

  s.dim=2
  s.vertices={[0;0],[1;1],[1;0]}
  s=makesimplex(s);

% See also PREPARESIMPLEX, DEMOSIMPLEX
```

# Chapter 27

# Configuration

The PAHSCTRL toolbox has a number of options that can be configured according to speed up calculations according to the specific problem or change e.g. simulation and plotting appearance. All configurations are added to the PAHSconf structure which should be placed in the workspace. Below is a sample configuration file showing the available config options.

example_conf.m

```
%Example configuration file for PAHStoolbox

% Should findBlockable start buttom op in order to reduce avg runtime?
% Most valuable on problems of high dimension
PAHSconf.blockqueueopt=true;

%Uncomment the below line to set a default seed for Monte Carlo sims
%PAHSconf.mcseed=1234;

%Set the default number of tests for Monte Carlo sims
PAHSconf.sims=15;

%Sets the default simulation time for Monte Carlo sims
PAHSconf.simtime=[0 0.4];

%User defined test vectors for finding control laws without fixed points
%PAHSconf.testVectors{1}=[0 1 0 0]'; %Here for a 4 dimensional state space
```

# Chapter 28

# The Algorithm

This part of the user's guide describes the control to facet algorithms which are the basis for control law synthesis for PAHS. For detailed descriptions of the theory which the toolbox is based on see the bibliography, section 33. The algorithm implemented in the toolbox consists of a four main tasks:

1. Compute LMIs describing the constraints
2. Determine a feasible controllable set of facets
3. Find a control-law in the feasible set
4. Make a graphic representation of the results

## 28.1 Prerequisits

The prerequisits for the algorithm is that the vertices of the state space simplex are known, the vertices/facets of the polytope describing the input space are known and the dynamics the affine dynamics of the system has been determined.

# Chapter 29

# LMI Constraints

There are a number of constraints arising from different requirements that needs to be computed as part of this algorithm:

- The input, $u$, is inside the input polytope
- The derivative $\dot{x}$ at the vertives $v_i$, $i = 1 \dots N$ is pointing in/outwards wrt. a facet.
- All derivatives in the same direction.

Several of these LMIs require knowning the normal to a facet and this can be computed as the null-space of a basis for the co-dimensional 1 space containing the facet.

$$n = null\left(\begin{bmatrix} v_2 - v_1 & v_3 - v_1 & \dots & v_N - v_1 \end{bmatrix}\right) \tag{29.1}$$

It is also important to know if the computed normal points inwards or outwards with respect to the polytope the facet is part of. This can be computed by knowing a point inside the polytope eg. a vertex that is not part of the facet in question.

$$s = -n^T(v_{inside} - v_1) \tag{29.2}$$

where $s$ is negative if $n$ is pointing inwards. The outwards pointing normal can then be computed as $n_{out} = sn$. The outward pointing normal for facet $i$ will hereafter be denoted $n_i$.

### Input inside the polytope

For the input, $u$, to be inside the polytope $u$ must be inside each of the polytopes facets. Computing a vector from a point on the facet to $u$ and then finding that vectors inner product with the facets normal, $n_i$ yields a scalar which sign determines if the point is on the inside or outside of the facet.

$$n_i^T\,(u - v_i) < 0 \tag{29.3}$$

### Derivatives pointing inwards

For the derivative to point inwards the inner product between the derivative and the outwards normal of the facet must be negative.

$$n_i^T\,(Av_i + Bu + C) < 0 \tag{29.4}$$

### All derivatives in the same direction

For all the derivatives to be in the same direction there must exist some vector $d$, such that.

$$d^T\,(Av_i + Bu_i + C) > 0\,, \forall i \tag{29.5}$$

This is not an LMI as $d$ and $u$ are both variable, but for fixed $d$, or $u$ the constraint can be tested. It is according to [Habets06a], section 33 possible to compute a set of LMIs testing this constraint, but it requires knowledge of the vertices of $u_i$. It might seem straigth forward to obtain the vertices as we know the input polytope, but the input at each vertice is further constrained by the requirement to block certain facets. There exists algorithms which should be able to compute these vertices, but they are not immidiatly accesible from matlab. The toolbox uses both normals of all facets as testvectors, $d$, along with any user defined testvectors specified in the configuration structure.

# Chapter 30

# Feasability checks

The main challenge of the computing discrete equivalents is finding feasible sets of blockable facets and determining whether each facet is controllable. This algorithm does not look for all possible sets of blockable facetes, but instead finds one blockable set with the maximum number of blockable facets for that simplex. It is then checked whether for each facet it can be guaranteed that it is controllable.

## 30.1 Blockable set

To find the largest blockable set all combinations of facets are tested starting with all facets blocked. Each tests consist of determining the feasability of the LMI-system consisting of the LMIs limiting the input to the input-polytope and a set of LMIs ensuring that the derivative points inwards, one for each vertex of the facets to be blocked.

1. For all facets of the input-polytope add an instance of the input LMI.
2. For all facets add $N$ instances of the inwards derivative LMI, one for each veretex of the facet.

## 30.2 Controllable facets

To determine if a facet is controllable it is necessary to establish weather a fixed point can be avoided in both configurations for the facet (opened and closed). The first check is for the closed configuration, where all facets are blocked. The fixed point check is done by finding a vector, $d$, in which direction the derivatives at all the vertices can point using the all derivatuves in the same direction LMI. It should be noted that the constraints used to determine blockability still has to hold, so the LMIs described in the previous section are added. There are no methods of uniquely finding this vector but at least the outwards normals to the blocked facets will not yield a result, and neither will convex combinations of them. As testvectors both the inwards and outwards normals to the facets are chosen as a feasible result will mean that the vector field in general points away from or towards one of the facets. If no control can be found which ensures fixed points outside the simplex the blockable set is no good and must be thrown away. Given that the closed configuration is feasible it needs to be established whether the individual facets are controllable. This is done by using the LMI system used for establising blockability, but reversing the derivative requirement on the facet under examination, $l$, i.e. changing $<$ to $>$ in the derivative pointing inwards LMI. Furthermore it is still necessary to search for a $d$ where all derivatives point in the same direction, but in this case there is a natural candidate vector, namely $n_l$, as satisfying the reversed constraints ensures that all but one of the derivatives at the vertices of the simplex points out of facet $l$.

# Chapter 31

# Finding a Control Law

Assuming that a discrete game has been constructed using the feasability checks described in the previous section and that a desired exit facet has been found for the simplex a control law needs to be found. Effectively this means that an input in the feasible set needs to be found for each vertex. To accomplish this some linear objective function needs to be specified, $J(s)$, with $J$ being a linear map. LMI solvers such as the one in Matlab can then find a feasible solution to the system of LMIs which minimises the objective function. Two such functions have been designed for the purpuses of this algorithm; one minimises the control input at any given instance in time while the other tries to maximise the probability of leaving via a specific simplex.

## 31.1 Minimizing the input at each time instant

As minimizing the input at each time instant is equal to minimizing the input at each point in the simplex, this can due to convexity be reduced to minimizing the input at the vertices. This can be achieved by the objective function $J = \sum u_i^T u_i$, as the input at each vertex is independent of the inputs at the other vertices. Obviously this is not a linear objective function but using an auxiliary variable and a Schur complement a linear equivalent can be found. For each $u_i$ minimizing $w_i$ subject to $u_i^T u_i < w_i$ will minimize $J$ i.e. $min(\sum w_i) \Rightarrow min(j)$ The inequality can be rewritten as:

$$u_i^T u_i < w_i \Leftrightarrow w_i - u_i^T u_i > 0 \Rightarrow \begin{bmatrix} w_i & u_i^T \\ u_i & 1 \end{bmatrix} > 0 \tag{31.1}$$

This is an LMI and can be added to the system of LMIs and the linear objective function $J_2 = \sum w_i$ can be minised to yield $u_i$ minimising $J$.

## 31.2 Increasing likelyhood of leaving through desired facet

To increase the likelyhood of leaving through a given facet, $i$, the derivative in the direction of that facet can be maximised $max\left\{\sum^j n_i^T(Av_j + Bu_j + C)\right\} = max\left\{\sum^j Bu_j\right\}$ with $j = 1 \dots N$ denoting the vertices of desired exit facet. Furhtermore the derivative in the direction of the unblockable facets, $k = 1 \dots P$, can be minimized $min\left\{\sum^{l(k),k} n_k^T(Av_{l(k)} + Bu_{l(k)} + \right.$ $\left. min\left\{\sum^{l,k} Bu_{l(k)}\right\}\right.$, with $P$ being the number of unblockable facets and $l(k) = 1 \dots N$ demotes the vertices of unblockable facet $k$. These to objective can be combined to$J = -\alpha\sum^j Bu_j + \sum^{l,k} Bu_{l(k)}$, which is a linear map in $u$, with $\alpha$ being a weighting parameter adjusting the desire to push towards the facet with the desire to keep away from unblockable facets . There is no proof available that this objective function yields the maximum likelyhood of leaving through the desired facet, but the compromise with $\alpha = 1$ seems reasonable, but it might yield better results to set $\alpha = 0$ as staying away from the uncontrollable facets will cause the state to leave the simplex via the desired facet.

## 31.3 Computing the control law

Using the methods from the previous sections to find the desired input at the vertices, $u_i$ an affine control law, $u = kx+g$, can be found. First notice that

$$\begin{bmatrix} u_1 & \dots & u_n \end{bmatrix} = \begin{bmatrix} k & g \end{bmatrix} \begin{bmatrix} v_1 & \dots & v_n \\ 1 & 1 & 1 \end{bmatrix} \tag{31.2}$$

for the affine control law to yield the desired inputs at the vertices. As shown in [bib.html this control law is unique and as a consequence $V = \begin{bmatrix} v_1 & \dots & v_n \\ 1 & 1 & 1 \end{bmatrix}$ is invertible yielding

$$\begin{bmatrix} k & g \end{bmatrix} = \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix} V^{-1} \tag{31.3}$$

# Chapter 32

# Visualisation

To make a graphical representation of a discrete equivalent the facets of the simplex is drawn with color codes representing the blockability. Possible derivatives are shown at the vertices, one for each of the input vertices and one for the barycenter of the input polytope. Furthermore Monte-Carlo simulations of the closed loop performance can be done using a uniform distribution of initial values inside the simplex

# Chapter 33

# Bibliography

**Habets06a** Habets, L. C. G. J. M.; Collins, P. J. & van Schuppen, J. H. Reachability and Control Synthesis for Piecewise-Affine Hybrid Systems on Simplices IEEE Transactions on Automatic Control, 2006, 51, 938-948

**Habets06b** Habets, L.; Kuut, A.; Nool, M.; Petreczky, M. & van Schuppen, J. H. ConPAHS - A software package for control of piecewise-affine hybrid systems Proc. 2006 IEEE Conference on Computer Aided Control System Design (CACSD.2006), 2006

**Grunnet08a** Grunnet, J. D.; Bak, T.; Bendtsen, J. D. & Larsen, J. A. Discrete Game Abstraction for Fault Tolerant Control Synthesis Proceedings of IEEE CACSD '08, 2008

**Grunnet08b** Grunnet, J. D.; Larsen, J. A.; Bak, T. & Wisnievski, R. A Piecewise Affine Hybrid Systems Approach to Fault Tolerant Satellite Formation Control Proceedings of the 3rd International Symposium on Formation Flying, Missions and Technologies, 2008

**Grunnet09a** Grunnet, J. D.; Bak, T.; Bendtsen, J. D. & Ankersen, F. PAHSCTRL - A Control Synthesis Toolbox for Piecewise-Affine Hybrid Systems, Proc. of 2009 ECC, Accepted

**Grunnet09b** Grunnet, J. D.; Bendtsen, J. D. & Bak, T.; Automated Fault Tolerant Control Synthesis based on Discrete Games, Proc. of 2009 CDC, Submitted

**Grunnet-ND** Grunnet, J. D.; Bendtsen, J. D. & Bak, T.; Automated Controller Synthesis for Piecewise-Affine Hybrid Systems, Submitted to International Journal of Control, Automation and Systems